APPENDIX A

Pseudo-Code for the Invention

```
/*      Load existing document database into memory.
        The data structure used is a hash with each hash value pointing to a balanced tree
        containing the ordered pair (Digest, DocId). The data structure is searched via the digest
        value.  */
DigestDB = LoadDocDB ( dbname ) ;

/*      Load the list of stop words to ignore and create a hash table.
        -- This step is optional if the user does not desire stop word removal        */
stopwordHash = LoadStopWordList (filename) ;

/*      Get a list of new documents to process.       */
DocsToProcess = GetDocsToProcess (processlist) ;

/*      Get first document to process.        */
DocToParse = DocsToProcess.nextDoc() ;

/*      Continue as long as there are documents to process  */
While ( DocToParse )
{
        /*      Create SHA1 Digest Object for current document      */
        SHA1 sha1 = new SHA1() ; //

        /*      Create Parser Object for current document     */
        Parser parser = new Parser(DocToParse) ;

        /*      The derived tree represents all the unique tokens from the current document.
                The tree is ordered in Unicode ascending order          */
        Tree docTokens = new Tree() ;

        /*      Continue iteration for as long as there are tokens to process */
        for ( ;; )
        {
                /*      Get the next token from the document          */
                token = parser.getNext() ;

                /*      If there are no more tokes to process, exit loop          */
                if ( token == null ) break ;

                /*      Using term thresholds, retain only significant tokens.
                        If parts of speech are used, remove the ignored parts of speech.
                        In the pseudo-code, only the removal of stop words are illustrated. If
                        other text components are to be removed, they should be removed at this
                        point.   */

                /*      Token is a stop word  */
                if ( stopwordHash.exists( token) == true ) continue ;

                /*      If there is a collision of tokens in the tree, only one is inserted.
                        For the current document, add token to tree of unique tokens         */
                docTokens .add ( token ) ;
        }

        /*      Create an iterator that traverses the tree of unique tokens defining of the current
                document        */
        Iterator iter = new Iterator ( docTokens ) ;

        /*      Loop through the tree of unique tokens for the document and add the token to the
                SHA object.   */
        for ( iter.GetFirst(); iter < docTokens.size(); iter++)
        {
                sha1.add ( iter.getValue() ) ;
        }

        /*      The computed digest value is created */
        sha1DigestValue = sha1.finish() ;

        if ( DigestDB.search ( sha1DigestValue ) )
        {
                /*      This is a similar document.  Print message and document name        */
                print ( "We have a duplicate document: %s", DocToParse.name() ) ;
        }
        else
        {
                /*      This is not a similar document.  Add to the collection          */
                DigestDB.add ( sha1DigestValue, DocToParse.name() ) ;
        }

        /*      Get Next Doc to process        */
        DocToParse = DocsToProcess.nextDoc() ;
}

/*      Write out the new document database to the file system        */
writeDocDB ( DigestDB, dbname ) ;
```